Implementation of an Ethernet-Based Communication Channel for the Patmos Processor

Pezzarossa, Luca; Kenn Toft, Jakob; Lønbæk, Jesper ; Barnes, Russell

Publication date:
2015

Document Version
Publisher's PDF, also known as Version of record

Implementation of an Ethernet-Based Communication Channel for the Patmos Processor

Luca Pezzarossa, Jakob Kenn Toft, Jesper Lønbæk and Russell Barnes

Department of Applied Mathematics and Computer Science
Technical University of Denmark, Kgs. Lyngby
Email: lpez@dtu.dk, [s113012, s094726, s146105]@student.dtu.dk

DTU Compute Technical Report-2015-02

June 1, 2015

Abstract

The Patmos processor, which is used as the intellectual property of the T-CREST platform, is only equipped with a RS-232 serial port for communication with the outside world. The serial port is a minimal input/output device with a limited speed and without native networking features. An Ethernet 10/100BASE-T IEEE 802.3 based communication channel is a reliable and high speed communication interface (10/100 Mbits/s) that also supports networking. This technical report presents an implementation of an Ethernet-based communication channel for the Patmos processor, targeting the Terasic DE2-115 development board. We have designed the hardware to interface the EthMac Ethernet controller from OpenCores to Patmos and to the physical chip of the development board, and we have implemented a software library to drive the controller and to support some essential protocols. The design was implemented on an Altera Cyclone IV FPGA in the aforementioned board, and it was tested with a software application, running on Patmos, that uses the Ethernet communication channel while the system is connected to a small local area network.

Contents

Chapter 1

Introduction

The T-CREST project developed a general-purpose time-predictable multiprocessor platform for embedded hard real-time systems [1]. The processor used as intellectual property in the T-CREST platform is the Patmos time-predictable processor [2, 3]. For the communication with the outside world, Patmos is equipped only with a RS-232 serial port [2]. The serial port is a minimal input/output device used for the 'stdout' and 'stdin' standard streams, with a limited speed (up to 115200 bits/s) and without native networking features.

The Ethernet 10/100BASE-T IEEE 802.3 protocol, which is a collection of specifications for layers one and two of the Open Systems Interconnection (OSI) model, provides the wiring and signalling specifications and defines the structure and the fields of the data frames to be received from or transmitted to the network. An Ethernet-based communication channel is a reliable and high speed communication interface (10/100 Mbits/s) that also supports networking. The implementation of an Ethernet-based communication channel for Patmos brings new challenges: from a hardware point of view, the main challenge concerns the integration of an Ethernet controller with the rest of the system, and from a software point of view, the challenges concern the development of a software library to interact with the Ethernet controller and to handle other higher networking protocols.

In this technical report, we present an implementation of an Ethernet-based communication channel for the Patmos processor, targeting the Terasic DE2-115 development board. The Ethernet-based communication channel is 10/100BASE-T IEEE 802.3-compliant and can reach a speed of up to 100Mbit/s. The implementation is split into two parts, a hardware part and a software part. The hardware part concerns the interfacing of the EthMac Ethernet controller from OpenCores to Patmos and to the physical chip of the development board. The software part provides primitive functions to drive the EthMac controller, to send and receive Ethernet frames, and the necessary functionality to support the following protocols: Ethernet-Medium

Access Control (Ethernet-MAC), Internet Protocol v.4 (IPv4), Address Resolution Protocol (ARP), Internet Control Message Protocol (ICMP), and User Datagram Protocol (UDP). From an OSI model perspective, the hardware part handles layer one and part of layer two (Ethernet-MAC), and the software part handles the remaining part of layer two and networking protocols of layers three and four.

In order to test the correct functionality and to evaluate the hardware resources utilization and performance of the design, we have implemented the system on an Altera Cyclone IV FPGA in the Terasic DE2-115 development board. The tests and the evaluation are carried out with a software application, running on Patmos, that uses the Ethernet communication channel to send and receive Ethernet frames and to manage IPv4, ARP, ICMP and UDP packages while the system is connected to a local area network.

This report is organized in 6 chapters: Chapter 2 provides the general background on the OSI model and on the protocols used in this work. Moreover, it presents related works. Chapter 3 introduces the Ethernet physical chip of the Terasic DE2-115 development board, presents the OpenCore EthMac controller and describes the hardware/software design and implementation. Chapter 4 describes the tests that were used to test and evaluate the design and reports the results. Chapter 5 briefly discusses some design features, states the contributions, and presents some possibilities for future developments. Chapter 6 concludes the report.

Chapter 2

Background and Related Work

The OSI model is a conceptual model used in computer networking that standardizes the functions of a communication system by partitioning it into seven logical layers. A layer serves the layer above it, and it is served by the layer below it. The layers of the OSI model, together with some protocols, are shown in Figure 2.1. The work presented in this report concerns only the hardware/software implementation of a set of protocols (Ethernet-MAC, IPv4, ARP, ICMP, UDP) that belongs to the four lowest layers. These protocols are underlined in Figure 2.1.

Network communication is based on encapsulation. Each layer of the OSI model includes, as a payload, the data provided by the layer above and adds some information (metadata) regarding the payload itself and networking-related informations (e.g., source and destination addresses, ports, etc.) according to the used protocols. The metadata are added as headers of the payload.

The Ethernet 10/100BASE-T IEEE 802.3 protocol is a collection of specifications for layers one and two (MAC) of the OSI model for wired Ethernet communication channels. For the Physical Layer (layer one), it defines the signalling specifications to transfer information via electrical pulses through a cable. This is done by a dedicated chip, called a physical (PHY) chip. It shuttles data back and forth from the cable to layer two and vice-versa with no regard to the information being transmitted and received.

The MAC (part of layer two) allows hosts of a network to communicate with each other, and it is also defined in the Ethernet specifications. Hence, we refer to it as Ethernet-MAC. The Ethernet-MAC is the first layer that performs actions based on the data. These actions include collecting data into frames and retransmitting chunks of data if issues occur. A protocol of layer two is the ARP. This protocol is used to discover the MAC address associated with a particular Network Layer (layer three) address. The encapsulation

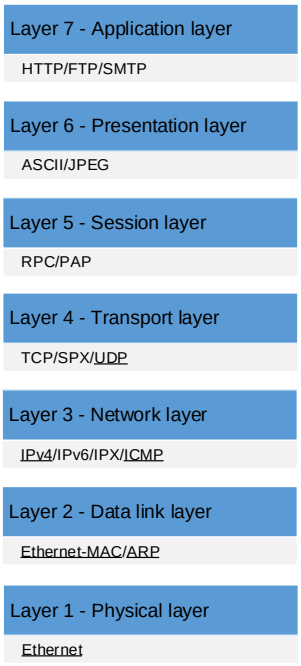
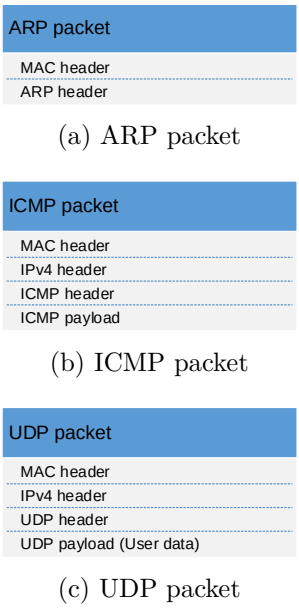


Figure 2.1: The layers of the OSI model; the underlined protocols are the ones used in this work.



(a) ARP packet

(b) ICMP packet

(c) UDP packet

Figure 2.2: Encapsulation structure of the ARP, ICMP and UDP packets.

structure of an ARP packet is shown in Figure 2.2a.

The Network Layer (layer three) adds another level of addresses in the payload. The Network Layer's addresses allow hosts that are not immediately connected at MAC level to communicate with each other, such as when packets pass through routers. A router inspects the headers of incoming packets and transmit them to the interface that corresponds to the correct address. The IPv4 and the ICMP protocols belong to this layer. The ICMP protocol is used by the hosts to perform tests and manage the traffic in the network (e.g., ping and traceroute operations). The encapsulation structure of an ICMP packet is shown in Figure 2.2b.

The Transport Layer (layer four) takes care of the reliable flow of information by adding further metadata to the datagrams. Metadata of this layer can, for example, keep track of the number and the order of packets received successfully. The UDP protocol belongs to this layer, and its encapsulation structure is shown in Figure 2.2c. The UDP protocol is a low-overhead, low-complexity transport protocol that does not attempt to guarantee the delivery of any individual packet.

Alternative designs to the one presented in this report can be found in both hardware and software sides. From a hardware point of view, Altera

proposes its Ethernet controller, called Triple-Speed Ethernet, and its integration method in [4]. The design provides a 10/100/1000 Mbps Ethernet MAC intellectual property that enables Altera FPGAs to interface to a PHY chip and create an Ethernet-based communication channel. From a software point of view, the FreeRTOS+UDP presented in [5] is a small, fully thread aware, sockets-based, UDP/IPv4 library for FreeRTOS. It is specifically designed for communication between small networks of embedded devices.

Chapter 3

Design and Implementation

The implementation of the Ethernet-based communication channel for Patmos is split in two parts, a hardware part and a software part. The following sections describe the target development board for this work, the Ethernet controller that was used, the hardware and software design, and its implementation.

3.1 Terasic DE2-115 Development Board

The Terasic DE2-115 development board is the target board for this project and it is equipped with an Altera Cyclone IV (EP4CE115) FPGA. With regards to the Ethernet communication, the board provides two Marvell 88E1111 Ethernet physical (PHY) chips and two RJ45 connectors. In this project, only one PHY chip and one RJ45 connector are used. As mentioned in Chapter 2, the PHY chip handles the layer one of the OSI model. It translates the data from layer two into signals to be sent in the Ethernet cable, and vice-versa. The Marvell 88E1111 PHY chip is 10/100BASE-T IEEE 802.3 compliant and can reach a speed of up to 1000 Mbit/s [6]. It is connected to the RJ45 Ethernet connector and to the FPGA, as shown in the block diagram in Figure 3.1. The interface between the PHY chip and the FPGA is the media independent interface (MII) that is a standard interface used to connect an Ethernet controller (layer two of the OSI model) to a PHY chip. A high speed version of MII, called Giga MII (GMII), that supports a communication speed of up to 1000Mbits/s is also supported by the chip, but is not used in this project.

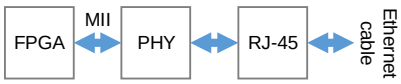


Figure 3.1: Block diagram of the PHY chip of the Terasic DE2-115 development board connected to the RJ45 Ethernet connector and to the FPGA.

3.2 EthMac Ethernet Controller

To provide an Ethernet-based communication channel to Patmos, a hardware Ethernet controller is required. For this project, we selected the open-source EthMac Ethernet controller from OpenCores [7, 8]. The main reason for choosing this Ethernet controller was the fact that it would allow code exploration and revision in case of difficulties with its integration with the system. Additionally, this also increase the portability of the system in other FPGAs, since the EthMac is not an FPGA-specific block (as opposed to the Ethernet controller provided for the DE2-115 by Altera [4]). The EthMac core is also 10/100BASE-T IEEE 802.3 compliant and can reach a speed of up to 100Mbit/s.

A simplified block diagram of the EthMac controller and its interfaces is shown in Figure 3.2. It is equipped with a slave Wishbone Bus (WB) interface, a master WB interface and a MII interface. WB is an open-source bus protocol for on-chip communication [9]. The slave WB interface is used to control the functionality of the controller and gives access to configuration registers and sending and receiving buffer descriptors. The master interface needs to be connected to a local memory, used to temporarily store outgoing and incoming Ethernet frames. In this report, we refer to this memory as the RX/TX buffer. Finally, the MII interface is used to drive the PHY chip.

The buffer descriptors are used by the EthMac core to manage the reception and the transmission of Ethernet frames. Each frame that has to be sent or received is associated with a buffer descriptor, and the buffer descriptors tell the core where to read or store the Ethernet packets. Each buffer descriptor consists of 64 bits; the first 32 bits are configuration and status flags, and the last 32 are used as pointers to the locations in the RX/TX buffer where the frames are stored when received or transmitted. The fields of a buffer descriptor are different depending on whether they are used for sending or receiving packets. The buffer descriptors are stored in a table within the EthMac controller. In total, there are 128 buffer descriptors. A configuration register defines how many descriptors are dedicated for transmitting, and the remaining ones are used for receiving. Not all buffer descriptors need to be used. For example, in the test presented in Chapter 4, two buffer descriptors are used, one for receiving and one for transmitting. Further information about the configuration registers, the functionality and the hardware design of the EthMac controller can be found in [7, 8].

3.3 Hardware Design and Implementation

The hardware design mainly concerns the interfacing of the EthMac Ethernet controller to Patmos and to the PHY chip of the development board. The Patmos processor provides an OCP protocol-based [10] interface for IO

devices. The Patmos interface used for this work is the 'OCPcore', and it has a data size of 32 bits and an address size is 16 bits [2]. As mentioned before, the configuration port of the EthMac controller is based on the WB protocol and has an address size of 12 bits. This requires some logic to translate the read/write transactions between the OCP and the WB and to properly manage the accesses to EthMac configuration registers, buffer descriptors and the RX/TX buffer.

Figure 3.3 shows a block diagram of the presented design. This is also the top level of the hardware implementation, and the development board associated PHY chip and connections can also be observed in Figure 3.3. The design consists of four blocks: the Patmos processor, the EthMac controller, the RX/TX buffer and a Demux/Bridge block.

The TX/RX buffer is a single-clock true dual-port memory implemented using Block-RAM (BRAM) of the FPGA. The memory port interfaced with

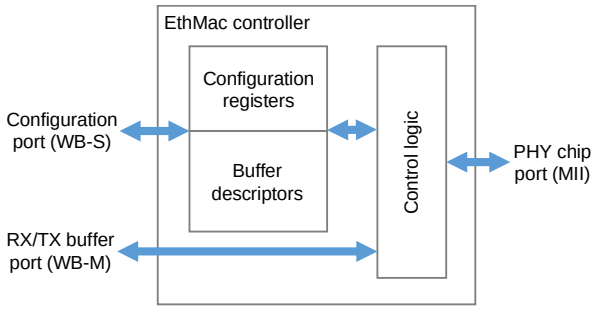


Figure 3.2: Simplified block diagram of the EthMac controller and its interfaces.

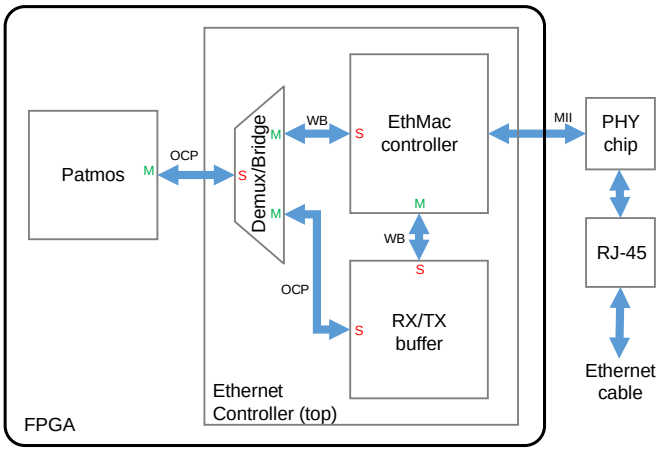


Figure 3.3: Block diagram of the hardware design of the Ethernet controller interfaced with Patmos.

Table 3.1: Division of the 16 bit Patmos address space into two sub-address spaces.

Patmos addr. space	Selected output
0000 - EFFF	RX/TX buffer
F000 - FFFF	EthMac port

Patmos supports the OCP protocol, while the port interfaced with the EthMac controller supports the WB protocol. The dual-port memory allows the EthMac to send and receive Ethernet frames and the processor to read and write frames in the memory simultaneously. The maximum dimension of the TX/RX buffer is 64 KB since the address size of the OCP interface is 16 bits.

The Demux/Bridge block manages the read/write transactions of Patmos to the EthMac controller and to the RX/TX buffer. Depending on the address, the read/write transactions of Patmos are sent to the EthMac configuration port or to the RX/TX buffer. When the read/write transactions are sent to the EthMac configuration port, they pass through an OCP/WB bridge, while the RX/TX buffer receives them directly since it supports the OCP protocol. The configuration interface of the EthMac controller has an address space of 12 bits, and it is located at the end of the address space of the Patmos OCP interface. Table 3.1 shows how the Patmos 16 bit address space is divided in two sub-address spaces for the RX/TX buffer and for the EthMac configuration port. When Patmos writes to or reads from the EthMac address space, only the 12 lower bits of the address are used. If the RX/TX buffer has a size of 64KB, the last 4KB are not accessible since the address space of the RX/TX buffer overlaps with that of the EthMac configuration registers and buffer descriptors.

3.4 Software

The software part consists of a library that provides primitive functions to drive the EthMac controller, functions to send and receive Ethernet frames and the necessary functionality to support the following protocols: Ethernet-MAC, IPv4, ARP, ICMP, and UDP. The functionalities provided by each section of the library are explained in the following subsections. All of the sections of the library provide functions to access the protocol-related fields of a received Ethernet frame.

3.4.1 EthMac functionality

This section of the library provides primitive functions to access the registers of the EthMac controller and other higher level functions. For example, to enable frame transmission and reception, functions are included for setting up buffer descriptors (both transmission and reception) and assigning a MAC address to the EthMac controller. When a MAC address is assigned to the controller, it will discharge packages if the destination MAC is not the assigned one nor a broadcast frame.

3.4.2 Ethernet-MAC functionality

This section of the library provides functions to send and receive an Ethernet frame. When an Ethernet frame to be sent is written to the RX/TX buffer, the send function sets up a buffer descriptor with all the needed information in order to start a sending operation. When an Ethernet frame is received, a flag in a configuration register is raised. This flag is polled by the receive function. The buffer descriptor for reception is then checked, and the RX/TX location for the received package is pulled. The type of package can then be checked, and depending on the type, different actions can be performed.

3.4.3 IPv4 functionality

This section of the library provides functions to access all of the IPv4 fields and provides help functions, such as the ones to compare IP addresses and to calculate and check the checksum. The functionality to manage the reception of fragmented packets is not provided by the library, and it is left to the user to achieve it using the IPv4-related functions of the library.

3.4.4 ARP functionality

This section of the library implements the ARP protocol and the ARP table-related functions. Mainly, it provides a function to reply to ARP requests and to send an ARP request in order to resolve an IP address. When an ARP package is received, the IP address to be resolved is checked, and if it matches the one assigned to Patmos, the reply function builds and sends back an ARP reply. The request function builds and sends an ARP request packet to resolve an IP address. This is used by protocols in higher layers in the OSI model, such as UDP. To avoid the need of resolving an IP address when a package is sent to the same IP address multiple times, an ARP table is implemented. The resolved IP addresses are stored here with their associated MAC addresses, and before sending an ARP request to resolve an IP address, a lookup is first made in the ARP table. Functions to manage the ARP table are also implemented (e.g., search an entry, remove an entry, clear the table, etc.).

3.4.5 ICMP functionality

This section of the library provides functions related to received ICMP packets, such as the function to answer ping requests. In the case that an ICMP ping request packet is received, a ping reply package is build and sent. Other functionalities of the ICMP protocol are not supported (traceroute, state of the network, etc.).

3.4.6 UDP functionality

This section of the library provides functions to access the UDP fields, such as the source and the destination ports, the payload length and the data in the payload. It also provides high level functions to send UDP packets. The send function takes, as arguments, the destination IP, the source and destination ports, the data length, a pointer to the data to be sent and a timeout value. It then checks if the IP/MAC association is known in the ARP table. If the MAC address is known, the function sends the packet; otherwise, it attempts to resolve the IP address with ARP. If the address is resolved before the timeout expires, the function sends the packet. Otherwise, it fails.

Chapter 4

Evaluation and Results

In order to evaluate the hardware resource utilization and test the correct functionality of the design, the communication channel was implemented on the Altera Cyclone IV FPGA (EP4CE115) on the Terasic DE2-115 development board.

4.1 FPGA Hardware Resource Utilization

Table 4.1 shows the FPGA hardware resource utilization in terms of logic cells (LC), digital signal processing elements (DSP), and bits of memory (RAM) of the entire platform and of some relevant entities of the design. The RX/TX buffer size is 64 KB, which is the maximum size achievable with a 16 bit address, and it is implemented using Block RAM (BRAM). The first rows of Table 4.1 report the hardware resource utilization of the entire system implemented on the FPGA. The second and the third rows report the resources needed for Patmos and the Ethernet controller (top). Finally, the last two rows report the resources devoted to the EthMac from OpenCores and the RX/TX buffer.

Table 4.1: Hardware resource utilization for the Altera Cyclone IV FPGA implementation of the design.

	Logic cells	DSP	BRAM (bits)
Entire system	12 503	8	660 992
Patmos	8 889	8	128 512
Eth. controller (top)	3 609	0	532 480
EthMac	3 464	0	0
RX/TX buffer	87	0	524 288

4.2 Testing of the Functionality

In order to test the correct functionality of the implemented design, we have developed a software application for Patmos that uses the Ethernet communication channel to send and receive Ethernet frames and to manage IPv4, ARP, ICMP, and UDP packages while the system is connected to a small local area network.

The test setup consists of the FPGA board connected with an Ethernet cable (cat 5e) to a personal computer (PC) running Ubuntu 14.04. The connection was made without a switch, router or hub in between the PC and the development board. Hence, the local area network has two hosts. The Ethernet connection is configured with a 10 Mbit/s speed in full-duplex mode. The PC and Patmos processor are both configured with static IP and MAC addresses. For the test, the IP and MAC addresses of the PC are 192.168.1.10 and 34:E6:D7:1A:92:48, respectively. The addresses of Patmos are 192.168.1.12 and 00:FF:EE:F0:DA:42.

The software application running on Patmos was developed to test the correct functionality of the design and of the functions provided by the software library with the following tests:

- Test 1 - Answer ping requests
- Test 2 - Drive development board LEDs over UDP
- Test 3 - Arithmetic calculations over UDP
- Test 4 - MAC address resolution using ARP

Patmos is also connected through the serial port with the PC, and the testing application uses this connection to print information related to the application status and the received packets (e.g., packet type, IP/MAC addresses, UDP ports, etc.). The software application Netcat was used to send and receive UDP packets on the PC. To further verify that the packets sent over Ethernet are correct, the software application Wireshark was used. Wireshark runs on the PC and captures all of the network traffic. For all of the tests, the captured data traffic was as expected, and all of the checksums were correct. The following subsections explain in detail the purpose of each test and presents the relative results.

4.2.1 Test 1 - Answer ping requests

The software library provides the functionality to answer a ping request (ICMP protocol). The testing application on Patmos checks this functionality by sending a ping request from the PC. This test was performed using the ping application available on Ubuntu 14.04 to send ping requests to the Patmos processor and read the results with regard to packet loss and response time.

Table 4.2: Patmos response time on ping request.

Payload size (bytes)	Min. time (ms)	Max. time (ms)	Avg. time (ms)
64	0.241	0.320	0.295
256	0.518	0.570	0.539
1024	0.926	0.997	0.974

Listing 4.1: Information of a received ICMP ping request packet.

```
- Level 2 protocol: Ethernet
- Level 3 protocol: IP
  - Source IP: 192.168.1.10
  - Destination IP: 192.168.1.12
  - IP checksum: E220 [OK]
- Level 3 protocol: ICMP
- Notes:
  - Ping to our IP, replied.
```

The minimum, maximum, and average response times of 30 ping requests for three different payload sizes are shown in Table 4.2. The results show that the latency is mainly due to the transmission of the packet itself. The packet loss for all of the tests is 0%. Listing 4.1 shows the information of a received ICMP ping request packet printed by Patmos through the serial port.

4.2.2 Test 2 - Drive development board LEDs over UDP

The purpose of this test was to verify that the Patmos processor is able to receive, decode and extract the data payload of UDP packets correctly. Hence, this test was constructed to test the one-way UDP communication (reception only) based on an UDP port selection. The test consisted of sending an integer number as a string of ASCII characters as payload of an UDP packet. Patmos decodes the string and drives the green LEDs of the board in a pattern matching the lower byte of the unsigned representation of the received number. For example, a value of 255 (0xFF) would turn the eight green LEDs on, and a value of 0 (0x00) would turn them off. When the UDP packet is received by Patmos, if the destination MAC and IP addresses match with those of Patmos, the IP and UDP checksums are correct, and the UDP destination port is the one dedicated to the LEDs test (1234 in our case), then the UDP data payload is extracted and the LEDs are driven consequently. This test proves that Patmos is able to receive an UDP packet, access its fields (i.e., the UDP ports), and use the payload data to perform actions. Listing 4.2 shows the information of a received UDP packet for the LEDs port printed by Patmos through the serial port.

Listing 4.2: Information of a received UDP packet for the LEDs port.

```
- Level 2 protocol: Ethernet
- Level 3 protocol: IP
  - Source IP: 192.168.1.10
  - Destination IP: 192.168.1.12
  - IP checksum: B305 [OK]
- Level 4 protocol: UDP
  - Source port: 36168
  - Destination port 1234
  - UDP checksum: 8913 [OK]
  - Data length 4 B
- Notes:
  - UDP packet to our IP and with UDP destination port for LEDs.
  - The UDP data is: 170
  - The extracted value is 170. The lower byte is   0xAA
```

4.2.3 Test 3 - Arithmetic calculations over UDP

This test can be considered an extension of Test 2 since it also verifies that Patmos is able to build and send UDP packets correctly. Hence, it tests the the two-way UDP communication (reception and transmission) also based on UDP port selection. This test consists of sending a string of ASCII characters from the PC that contains an arithmetic operation (sum, subtraction, multiplication and division) on integer numbers as the payload of an UDP packet (e.g. 20 + 108, 5 - 37, -12 * 5, 1024 / -32, etc.). When the UDP packet is received, if Patmos is the destination, the IP and UDP checksums are correct, and the UDP destination port is the one dedicated to arithmetic calculations (1235 in our case), then Patmos decodes the operation, executes it and sends an UDP packet back to the PC that contains a string of ASCII characters with the result of the expression. This test ensures that Patmos can build and send UDP packets using the implemented functions for UDP. Listing 4.3 shows the output of Netcat application running on Ubuntu used to send and receive UDP packets with the operations and the results, respectively. Listing 4.4 shows the information of a received UDP packet for the calculation port for the first operation required (20 + 108).

Listing 4.3: Output of the Netcat application used to send and receive UDP packets.

```
$ nc -4 -u 192.168.1.12 1235
20 + 108
Answer from Patmos: 20 + 108 = 128
5 - 37
Answer from Patmos: 5 - 37 = -32
-12 * 5
Answer from Patmos: -12 * 5 = -60
1024 / -32
Answer from Patmos: 1024 / -32 = -32, R = 0
```

Listing 4.4: Information of a received UDP packet for the calculation port.

```
- Level 2 protocol: Ethernet
- Level 3 protocol: IP
  - Source IP: 192.168.1.10
  - Destination IP: 192.168.1.12
  - IP checksum: 0219 [OK]
- Level 4 protocol: UDP
  - Source port: 32847
  - Destination port 1235
  - UDP checksum: 4A7E [OK]
  - Data length 9 B
- Notes:
  - UDP packet to our IP and with UDP destination port for
    ↪ calculation.
  - The UDP data is: 20 + 108
  - Received a valid string, replied.
```

4.2.4 Test 4 - MAC address resolution using ARP

The purpose of test number one verifies the ARP protocol and ARP table-related functions of the library. The MAC addresses of the PC and of the development board are both unknown to the other host of the local area network. Therefore, to receive from and send to other hosts in the the network, they must be resolved using the ARP protocol. The fact that Patmos is able to answer ping requests and receive UDP packets proves that it is able to answer ARP requests from other hosts. Listing 4.5 shows the information of a received ARP request packet printed by Patmos through the serial port. The request was replied by Patmos, and this was also verified with Wireshark. Analogously, the fact that Patmos is able to send UDP packets to other hosts in the network proves that it is able to send ARP requests, receive and manage the corresponding response and correctly store the associated IP and MAC addresses in its ARP table. Listing 4.6 shows the first entries of the ARP table after the IP/MAC address resolution. The first entry of the table contains the IP and the MAC addresses of the PC obtained using the ARP protocol.

Listing 4.5: Information of a received and replied ARP request packet.

```
- Level 2 protocol: Ethernet
- Level 3 protocol: ARP
  - Sender MAC: 34:E6:D7:1A:92:48
  - Sender IP: 192.168.1.10
  - Target MAC: 00:00:00:00:00:00
  - Target IP: 192.168.1.12
  - Operation: 1 [request]
- Notes:
  - ARP request to our IP, replied.
```

Listing 4.6: ARP table after IP/MAC addresses resolution.

```
ARP IP/MAC table
#   Used   IP              MAC
0   1      192.168.1.10    34:E6:D7:1A:92:48
1   0      0.0.0.0         00:00:00:00:00:00
2   0      0.0.0.0         00:00:00:00:00:00
[...]
```

Chapter 5

Discussion

The Ethernet-based communication channel provides Patmos with the ability to send data to and receive data from other hosts over a network. The results reported in Chapter 4 show the correct functionality of the design when implemented on the Terasic DE2-115 development. However, the design can be considered compatible with all boards provided with a PHY chip that uses the MII interface. The overall hardware resource utilization of the implemented Ethernet controller is relatively limited (about 40% of the Patmos one), especially considering its usage in a multi-core platform where only one processor is equipped with the controller.

In summary, the contributions of the work presented in this report to the T-CREST project are:

- We have integrated the OpenCores EthMac controller to Patmos targeting the Terasic DE2-115 development board and, in general, systems compatible with the MII PHY chip interface.
- We have provided primitive software functions to drive the EthMac controller.
- We have provides a software library with functions related to Ethernet-MAC, IPv4 and UDP protocols, managing the ARP protocol and the ARP table and replying to ICMP ping requests.

The preliminary version of the system used two separate WB interfaces to communicate with the EthMac controller and RX/TX buffer, so an OCP/WB bridge was developed in Chisel to equip Patmos with a WB interface. This is no longer used, but it can also be considered a minor contribution.

The software library is developed in a very structured way. This allows it to be easily expanded in the future with other protocol implementations and it gives the possibility for the porting of open-source libraries. For example, the software support for the File Transfer Protocol (FTP) or the Trivial File Transfer Protocol (TFTP) for reliable communication would allow files to be

uploaded to and downloaded from Patmos over a network. This can be used to speed up the programming phase of Patmos that is currently performed through the RS-232 serial port.

Chapter 6

Conclusion

In this report, we have presented an implementation of an Ethernet 10/100BASE-T IEEE 802.3 based communication channel for Patmos, targeting the Terasic DE2-115 development board. This includes the integration and the interfacing of the EthMac Ethernet controller from OpenCores with Patmos and the physical chip of the development board, a software library that provides primitive functions to drive the EthMac controller, and support for the Ethernet-MAC, IPv4, ARP, ICMP, and UDP protocols. The design was implemented on an Altera Cyclone IV FPGA in the Terasic DE2-115 development board, and its correct functionality was tested with a software application running on Patmos that uses the Ethernet communication channel to send and receive packets (e.g., drive development board LEDs and arithmetic calculations over UDP) while the system is connected to a two-host local area network.

Bibliography

[1] Martin Schoeberl, Cláudio Silva, and André Rocha. T-CREST: A time-predictable multi-core platform for aerospace applications. In *Proceedings of Data Systems In Aerospace (DASIA 2014)*, Warsaw, Poland, June 2014.

[2] Martin Schoeberl, Florian Brandner, Stefan Hepp, Wolfgang Puffitsch, and Daniel Prokesch. Patmos reference handbook. Technical report, 2014. [Online.] Available: `http://patmos.compute.dtu.dk/patmos_handbook.pdf` (visited on May 20, 2015).

[3] Martin Schoeberl, Pascal Schleuniger, Wolfgang Puffitsch, Florian Brandner, Christian W. Probst, Sven Karlsson, and Tommy Thorn. Towards a time-predictable dual-issue microprocessor: The Patmos approach. In *First Workshop on Bringing Theory to Practice: Predictability and Performance in Embedded Systems (PPES 2011)*, pages 11–20, Grenoble, France, March 2011.

[4] Altera corporation. Triple-speed ethernet megacore function. Technical report, 2014. [Online.] Available: `https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_ethernet.pdf` (visited on May 20, 2015).

[5] FreeRTOS+UDP webpage. [Online.] Available: `http://www.freertos.org/FreeRTOS-Plus/FreeRTOS_Plus_UDP/FreeRTOS_Plus_UDP.shtml` (visited on May 20, 2015).

[6] Marvell. 88E1111 product brief. Technical report, 2013. [Online.] Available: `http://www.marvell.com/transceivers/assets/Marvell-Alaska-Ultra-88E1111-GbE.pdf` (visited on May 20, 2015).

[7] Igor Mohor. Ethernet IP core specification. Technical report, 2002. [Online.] Available: `http://opencores.org/project,ethmac` (visited on May 20, 2015).

[8] I. Mohor. Ethernet IP core design document. Technical report, 2002. [Online.] Available: `http://opencores.org/project,ethmac` (visited on May 20, 2015).

[9] ORSoC and OpenCores. Wishbone B4 specification. Technical report, 2010. [Online.] Available: `http://cdn.opencores.org/downloads/wbspec_b4.pdf` (visited on May 20, 2015).

[10] OCP International Partnership. Open Core Protocol specification - release 3.0. Technical report, 2012. [Online.] Available: `http://accellera.org/downloads/standards/ocp/` (visited on May 20, 2015).